

Exploiting Model Driven Technology

A Tale of Two Startups

Tony Clark¹ and Pierre-Alain Muller²

¹ Middlesex University, The Burroughs, Hendon, London NW4 4BT, UK t.n.clark@mdx.ac.uk

² Universite de Haute-Alsace 12, rue des Freres Lumiere F-68093 Mulhouse Cedex France
pierre-alain.muller@uha.fr

Abstract. This article describes the experiences of two independent start-up companies that were created in the white-heat of the early days of model based engineering. Each company aimed to revolutionise software development by raising the level of abstraction through modelling. The article describes the context, technical innovations, business experiences, demise and lessons learned by each company.

Keywords: Startup, Tool Company, Model Driven Development.

1 Introduction

“It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair, we had everything before us, we had nothing before us, we were all going direct to heaven, we were all going direct the other way.” [A Tale of Two Cities, Charles Dickens (1859)]

A revolution in Software Engineering occurred in the late 1990s with the rise of UML and associated Model Driven Development (MDD) approaches to software development. At this time many disparate threads of object-oriented design were fused into a single unified notation. Tools that supported the single notation started

to emerge and the risk of technological obsolescence reduced to the point where many companies felt able to adopt UML as part of the construction and maintenance of software systems.

The research community saw an opportunity to contribute to this technology wave by formalizing and extending UML in various ways. Since UML was adopted so widely by industry, the opportunity to deploy high-technology results from these research activities became irresistible. The MDD approach to software development promised to revolutionize how systems are created and maintained. Technology evangelists spread the word to whoever would listen. Several startup companies were formed in order to promulgate their founders’ technological revolutionary zeal.

This article describes the experiences of two high-technology MDD startups that were formed independently by the authors as part of the whirlwind of activity that occurred at this time. We briefly review the key features that led to the creation of the companies in section 2 and then give an overview of their creation and eventual demise in section 3. Finally we review some of the key lessons that can be drawn from the experiences of both companies in section 4. Whilst the article aims only to present the personal experiences of the authors, we have attempted to place these in the context of a typical technology wave by relating them to a spin-out life-cycle. Our contribution is to document the experiences and lessons learned typical to high-tech startups, and to provide a historical record of two attempts to exploit model-driven development as it was emerging.

2 The MDD Landscape

The two companies described in this article cover the ten years from around 1998 to 2008. In order to place the histories of the companies in context, this section describes key trends and events that relate to the development of the MDD landscape:

1980s The 1980s saw the introduction of object-oriented (OO) programming languages including Smalltalk, C++ and Objective-C.

1990s OO Development methods started to emerge and to replace structured methods. At this time there were many different meth-

- ods all of which had different notations and proposed different life-cycle steps.
- 1997** Around 1997 several competing OO methods were merged to become the Unified Modeling Language (UML). The single notation and significant commercial backing made the notation attractive to tool vendors. The Rational Rose toolkit became the first to support UML and achieved strong sales across the world. At this time, Rational Rose was essentially an editor for constructing UML diagrams.
- 2000** UML versions 1.x were generally viewed to be too small-scale to represent large systems. The Object Management Group (OMG) initiated the design of UML 2.0 that would see a considerable increase in notational size and complexity. In addition, the OMG introduced Model Driven Architecture (MDA) [30,18,23], a way of using UML to transform from high-level models to lower-level implementations (essentially code generation). The OMG introduced the notion of the *Meta-Object Facility* (MOF) as a means to describe the relationships between various elements of a modelling language architecture. The architecture consists of 4 levels: the language definition level (M3) that is used to define languages, MOF is defined at this level; the modelling language level (M2) where languages are defined as instances of M3-models, UML is defined at this level; the model level (M1) that contains user-models as instances of M2-models, all UML models exist at this level; the instance level (M0) that contains occurrences of elements that are instances of M1-models. MOF and the 4-level architecture was important for two reasons: extensibility of UML and interoperability. Extensibility was defined in terms of additions to the definition of UML at level M3 (manipulated as instances of M4-models). Interoperability was defined in terms of a serialization format defined in terms of M4-features.
- 2001** The Eclipse toolkit had been developed in the late 1990s and started to emerge as a tool platform for software systems engineering. It was designed with a plug-in architecture to allow multiple tools to co-exist and to co-ordinate. Eclipse could be used to develop commercial tools, however the Eclipse Public License was developed to encourage open-source collaborative development. Graphical libraries on Eclipse could be used to develop modelling tools.
- 2002** To Support MDA the OMG initiated the development of queries views and transformations on models. This led to the QVT standard and to other open-source technologies to implement model transformation and code generation from models. Around this time several modelling tools emerged that started to compete in the market-place. The open-source tools generally supported limited modelling (often class-based) which matched the predominant industrial use of modelling.
- 2004** The Eclipse Modeling Framework (EMF) emerged as an open-source Java based technology for implementing modelling languages [31].
- 2006** The Eclipse Graphical Modelling Framework (GMF) emerged as an open-source Java based technology for implementing graphical modelling languages and their associated editors.
- 2008** By this stage hundreds of modelling tools had become available. Some tools were commercial, but many were open-source. Some tools were general purpose and some were linked to specific technologies, for example to generate code for Java or C++.
- 2012** Currently model-driven approaches to system development are used in a wide variety of commercial application areas including telecomms, automotive, and mobile phones industries. They are used throughout the development life-cycle from requirements through to executable models for embedded systems. A recent study [16] reports a high degree of continued support for MDE amongst practitioners despite the lack of hard evidence (over 10 years after the field erupted) to support the claim. The study goes on to state: *Even allowing for a reasonable number of tool vendors in the survey sample, [...] indicates a significant level of disaffection with the tools used for MDE alongside a belief that the tools on offer are too expensive.*

3 Startups and Shutdowns

The previous section has described the MDD landscape during the period 1998-2008. This section describes the experiences of two MDD

startups during this period from startup to shut-down. In each case we give the company background, the technical basis for the company, the commercial experiences of the company and the reasons for its demise. The following section will analyse the company histories and propose a collection of lessons that can be learned.

3.1 Critical Junctures

There is little research into high-tech university spinout companies (USOs), however the authors of [32] report on phases of USO growth and the associated critical junctures (reproduced in figure 1). The phases of growth are:

Research The commercial basis for a USO emerges from academic research that is typically motivated by a *publish or perish* mentality.

Opportunity Framing The commercial opportunities of the research are identified, usually in conjunction with non-academic input.

Pre-Organization A business plan is created that targets specific markets. The knowledge and resources necessary to bring the research to market are identified. Seed funding is acquired.

Re-orientation Continuous reconfiguration of a commercial offering based on the research until a product or products of value to customers is identified.

Sustainable Returns The USO has achieved traction in the marketplace and substantial revenue streams have been established.

The associated critical junctures occur between successive phases. The junctures must be successfully negotiated in sequence if a USO is to become commercially sustainable:

Opportunity Recognition Breakthrough ideas are captured that can trigger an evaluation as a precursor to the formalization of a commercialization effort. [29]

Entrepreneurial Commitment The ideas and intentions of an entrepreneur play a critical role in the success of a venture.

Credibility The commercial basis for the exploitation of research must be commercially credible. Sources of knowledge and resource must be sufficiently convinced to participate. It is usual for software startups to rely on venture capital funding [21].

Sustainable Returns In order to negotiate the final hurdle, a USO must establish sustainable returns in the form of sales channels, milestone payments, or further investment.

3.2 XMF-Mosaic

Background Around 1999 the OMG started to think about UML 2.0 in order to rationalize and extend UML 1.x. At that time the Object Constraint Language (OCL) had joined the UML family. The pUML group had started to promote the idea of a ‘precise’ UML and there was interest from academic researchers in contributing to the new UML 2.0 initiative.

Some of the pUML researchers started to attend the OMG meetings and to contribute to discussions. Bran Selic and Steve Cook were prominent members of the OMG and were keen on developing UML 2.0 as a family of languages. This idea was taken up by Tony Clark, Andy Evans, Stuart Kent (CEK) and others who were supported by Bran, Steve and others to produce a proposal showing how UML 2.0 could be developed as an extensible language family [8].

Work by CEK led to an approach for model-based language engineering [5,4,3,7]. This approach was based on models for languages and their denotations with modelled relationships between them. This tripartite approach: syntax; denotations; semantics, could be used to explain a variety of languages including text-based and graphics-based.

The approach to language definition was developed in 2000-2001 and was validated by implementing a tool called MMT [6] that supported a template mechanism for language modules and their instantiation. MMT was written in Java and supported a graphical user interface for modelling similar to Rational Rose extended with a text-based language. The interesting thing about MMT that continued throughout its subsequent evolution was its basis in languages such as Lisp and Smalltalk and its relationship to OCL.

At this stage the OMG was developing ideas that came to be called Model Driven Architecture (MDA). Clark and Evans (CE) came to the conclusion that Model Driven Development (MDD) and MDA could not be fully realised without some key technology features: executable modelling, meta-modelling, language-

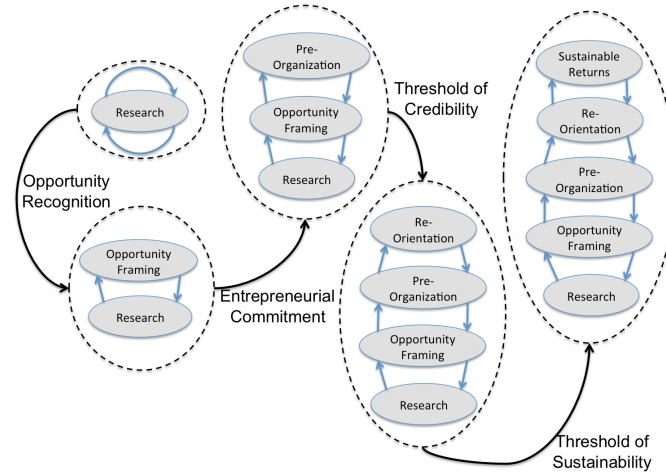


Fig. 1. Critical Junctures for University Spin-Outs [32]

engineering; and that a tool supporting these technologies would lead the way.

Unfortunately, MMT had been developed to support the UML 2.0 work and was too bound up with package instantiation and merge mechanisms that were needed to define a static modelling language. CE wanted the ability to program in terms of models and then to execute the models. A new initiative was started that aimed to generalize MMT so that its core concepts were meta-modelling and executable modelling.

XMT was a refinement of MMT that implemented xMOF (a proposal for an executable MOF) and the initial proposal for QVT [19]. Since Xactium’s background was in a variety of languages with meta-facilities including CLOS [2], ObjVLisp [11] and Smalltalk [14], it understood how to define a small extensible executable meta-kernel. Such a meta-kernel was important because it supported arbitrary extensibility compared to the limited extensibility provided by MOF. Whilst the UML world was fixed to a 4-level architecture rooted at MOF (a static language), XMF could support any number of levels rooted at XCore (a dynamic language). Like the designers of Smalltalk, arbitrary extensibility was viewed as being essential for defining languages and their supporting tools. However, unlike the meta-kernel of Smalltalk, XCore meta-classes were not limited to having a single instance. Therefore, language features could be factored into single meta-classes that were shared by multiple classes.

XMT was also designed to include plug-ins because of recent exposure to Eclipse and its extensible framework for tailored tools. XMT started to provide a basis for extensible syntax, like lisp-macros, but did not provide a mechanism for programming/modelling graphical languages. XMT was implemented as a Swing application.

Startup Around this time CE had become Model Driven Development (MDD) evangelists and had interest from several organizations including a large avionics company who were interested in using MDD on a particular project. They were convinced that MDD would be the right approach to make progress and that the XMT toolset could be used to do this. Their investment in the technology allowed Xactium to be founded and CE as co-founders to leave their respective academic jobs.

Technical Rationalization In 2004 Eclipse was starting to become well-known. Two researchers joined Xactium, bringing the staff number to four. Their experience and enthusiasm for Eclipse, together with the rather academic-looking state of XMT, led to the decision that XMT should be ported to Eclipse.

The Eclipse port afforded the opportunity to review XMT and the possibilities for MDD. It was realized that whilst XMT provided a vanilla class-based graphical modelling experience, people were starting to be interested in graphical domain specific modelling. In addition, it

was argued that any technology supporting true MDD needed to be able to be an extensible language workbench, and therefore by extension needed to be completely reflexive.

In 2004/5 XMT was completely rewritten for Eclipse and became XMF [20,9,1]. The key features of XMF were: an executable meta-kernel (XCore); an executable version of OCL including higher-order functions (XOCL); a byte-coded VM in Java that supported meta-object protocols (MOPS); model transformations; monitors on objects (daemons); threads; garbage collection; a foreign function interface to Java; a code template mechanism; a graphical library for Eclipse (MOSAIC) supporting browsers, diagrams and editors; various import-export formats including a language for parsing XML. The compiler and run-time system were completely boot-strapped so that the only things not written in XMF-Mosaic were the low-level graphics primitives and the VM. In particular, most technical features of the tool were written as embedded DSLs.

Commercialization Xactium tried to generate business based on XMT and XMF. Business with the Avionics company continued along with small-scale projects with several major companies, e.g. [13]. Commercial sales of the tool were small with prices that approximated those of Rational Rose. It became increasingly clear that customers viewed XMF as very leading edge technology and not commercially realistic. Xactium was viewed as short term consultants but not as commercial suppliers. The Xactium sales team was exclusively technical with no commercial experience.

Around 2004 the market for such tools became much smaller since tools were becoming available open-source. The market was disappearing and the Xactium directors could not agree whether the product should be open-source or not. Xactium had spent over a year exclusively in product development, cushioned by a handful of customers and had not spent much effort raising the profile of the company and technology through conferences and personal visits.

Raising Money In 2005 it became clear that Xactium needed to raise more funds. It also became clear that there was a strong technical direction to the company, but commercial exper-

tise was very weak. This imbalance caused friction that the company did not recover from as things went from bad to worse.

Xactium directors visited TogetherSoft (recently acquired by Borland) offices in Prague in the summer of 2005. The visit led to a recommendation from the TogetherSoft technical management to buy Xactium. In the end, a concrete offer did not materialise, however this strengthened Xactium's belief that it had a very strong technical offering.

The conviction that XMF was right for MDD was as strong as ever and various small-scale activities were initiated to raise funds. Regional development agencies (government business development grants) of various types were approached with some success. One such organization had set itself up as a form of venture capital based on a grant with 50% of the funds raised from high net-worth individuals.

Unfortunately, given the nature of such grants, the organization was not a specialist in the market and was not in a position to offer practical help. After a few attempts, Xactium was successful in raising approx 500K of capital with various strings attached. One condition was to take on commercial expertise since it was clear that Xactium had technical expertise but no idea how to monetize it.

In 2006, and after several false starts, Xactium recruited a commercial manager with a background in a relevant area. The investment also allowed Xactium to take on people with financial expertise and an office manager. It was clear that an injection of commercial expertise was essential for Xactium to continue.

The commercial manager was chosen because of long experience with software startups. Unfortunately, this turned out to be fatal since the individual concerned made no attempt to develop any business for the company and started to play politics with existing staff. A lack of *hunger* for making the business a success seemed to be an influencing factor.

A Commercial Saviour? In 2005/6 Xactium used XMF-Mosaic to implement a tool for a new method being trialled by a large US bank. This led to some funded development work and a large contract being negotiated in early 2007. However, the expectations of the customer were not met by the current state of the technology. In addition the size of the company (3 de-

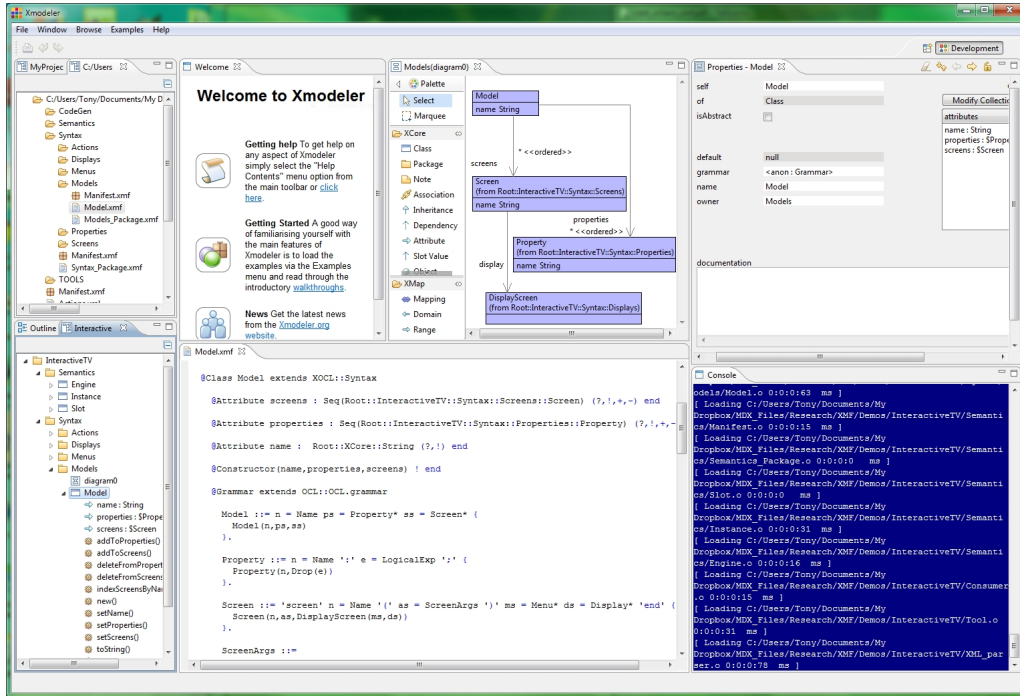


Fig. 2. The XModeler Tool

velopers and 2 commercial managers) did not give sufficient confidence regarding support and longevity. The Bank issued the contract and then rescinded it the following day citing internal reorganization (a sign of the looming financial crash?).

Shutdown At this point Xactium had put all of its commercial eggs into a single basket. It claimed to be a MDD company with a product. Unfortunately, the amount of investment in the product and therefore the size of the company made it almost impossible to do business of sufficient value despite being independently evaluated as the leading software engineering development platform [15] in terms of support for levels of abstraction.

The company directors were unable to agree how to continue and, as is typical in small startups when things go wrong, positions became entrenched and progress became difficult. One option was to retain the technical basis upon which the company had been started and to aim to partner with a large company (or even be acquired) effectively becoming a branch of a research lab. Another option was to completely change direction and use any remaining funds

to start a new business. The former was not really commercial and the latter was at odds with the basis on which the company had been created in the first place not to mention lacking any coherent focus.

The company eventually split in 2008, with all but one of the early-stage members leaving. The remaining director adopted the company name for a commercial endeavour unrelated to model-driven technologies after having tried several different market offerings. This is perhaps an extreme example of the claims made in [24] that startups should be managed through a succession of exploratory projects.

All outgoing members have continued with model-driven technologies in one form or another in a professional capacity. XMF and XMF-Mosaic (now called XModeler) is available under the Eclipse Public License³. It is shown in figure 2 with sub-panels shown clockwise from top-left: a file browser; a model-integrated web-browser; a class diagram editor (an instance of a user-extendible diagram tool meta-model); a property editor (an instance of a user-extendible property editor meta-model); a command console (showing the output after load-

³ <http://www.xmodeler.org/>

ing XOCL text files); the XOCL definition of a model (including a textual DSL for a modelling language); a model browser (an instance of a user-extensible browser tool meta-model). An overview of XModeler capabilities is described in [10].

Conclusion A key reason for Xactium’s inability to progress from startup to a stable company was its lack of commercial expertise from the outset. The founding members were too focussed on technology and lacked the flexibility and commercial acumen to achieve sufficient regular income. However, this is probably not the only reason for commercial failure. Model-driven technologies are abstract and general in nature, and incomplete because of their immaturity. A market for MDD tools probably did not exist at the time the company was set up, despite all the hype and enthusiasm. There are a number of current success stories of small companies with technology and individuals who evangelise leading edge technologies including MDD; however, they do not appear to scale which indicates that their revenue streams are perhaps not based on technology sales but on know-how. It would also not be unrealistic to suggest that such companies are existing on income from mundane software development that is used to fund more interesting activities.

In terms of the *critical junctures* described in section 3.1, Xactium based its *opportunity recognition* on the hype surrounding MDA and a single successful engagement with a large company. This, together with *entrepreneurial commitment* from established academics from two different institutions, was used to establish funding from a variety of sources that was necessary to negotiate the *threshold of credibility*. However, it could be argued that this was achieved prematurely and Xactium fell into a *re-orientation* cycle from which it never recovered.

The last decade has seen a number of high profile commercial MDD tools fail; recall Optimal-J or XDE? These tools were very expensive to produce and maintain, probably contributing to their demise. There are a number of open-source success stories, but it is not clear that these systems can support a business model. There have been some suggestions that many of the tools that failed, were based on UML, and that it is likely that if MDD tools succeed in the future then they are more likely

to be based on domain-specific modelling languages. This may be true, however there is no clear evidence to support the claim either way.

It seems reasonable to suggest that there is no significant MDD tools market currently waiting to be exploited and that it is more likely that MDD has started to become fragmented and embedded within more traditional approaches where it makes sense.

3.3 ObjeXion

Background In June 1998 Pierre-Alain Muller and Jean Bezivin created the UML series of conferences, originally intended to bring together academics and practitioners, around the emerging modeling language UML. Clearly, UML was becoming a hot topic in the software engineering community.

At that time, Pierre-Alain Muller who had been consulting and mentoring on object-oriented methods (Booch, OMT and then UML) during the previous decade, felt that a model-level prototyping tool would greatly help engineers to build models, and started to discuss this idea with potential users. The idea was well received and several companies agreed to beta-test the tool once it was available.

Startup Tool development was initiated as an add-on of Rational Rose. The idea was to tightly integrate the prototyping tool with the modeling tool to facilitate round-trip engineering. In early 1999 a new company was created and a marketing partnership was created with Rational Software (the owners of Rational Rose). The agreement allowed ObjeXion to resell Rational Rose, and also allowed Rational sales teams to propose the prototyper as an add-on to Rational Rose. ObjeXion Software acquired seed funding (about 250K euros) and accelerated technical development. The team grew up to 5 software engineers, and mid-1999 a sales person joined the team and started to develop a marketing strategy.

During the summer of 1999, a first version of the *model prototyper* was available and beta-testing was conducted with an avionics company. Three other companies that had originally accepted to be part of the beta program were either no longer available or unable to devote enough attention to make the beta worthwhile. Having only one beta site turned out to be a significant disadvantage because it gave too much

weight to one specific point of view. This led the team to develop specific tool functionality from a single perspective that was not of general interest.

Commercialization and Rationalization

The business model was based on selling software licences where the price of a single license was similar to that of Rational Rose. Despite a lot of commercial efforts the tool sold very few copies. Certainly sales figures were not enough to generate sustainable revenues nor to convince potential investors that more marketing would produce more revenues.

The marketing partnership with Rational was not as efficient as originally expected. There was a double problem with this partnership. Firstly, customers were not inclined to pay as much for an add-on as for the base product; secondly, customer's budgets tended to be fully consumed by Rational Rose tokens, leaving little opportunity for buying add-ons.

By the end of 1999, it was clear that the original business model was not realistic. After several discussions with business angels it was decided to develop a standalone product that would be able to go beyond prototyping and that would not require prior acquisition of Rational Rose.

However, the model-prototyping technology had been optimized for prototyping and was not suitable for production code. A lot of effort had been put into model evolution (for example migrating instances as much as possible when classes are modified) but not for execution speed, nor for scalability.

Therefore the team decided to refactor the technology, and to move from model interpretation to code generation. It was decided to develop in Java for the IDE and the front end and to use XSLT for code generation via model transformation. Experience showed that this was not very practical. Over time the XSLT components were deprecated and more and more parts of the back-end were developed in plain Java.

Technical Features Netsilon (shown in figure 3) [28,27] is a standalone tool for model-driven web engineering with UML class diagram modeling facility (borrowed from Argo UML), executability (Xion language that is a mix of Java and OCL), and model to text generation

(a DSL for HTML code generation). A lot of effort was put into infrastructure functionality (for instance persistence) since at that time the Eclipse platform was not available.

Netsilon implements the *Platform Independent Model* (PIM) vision. The company designed a persistent language named Xion based on UML for its structural part and an action language based on Java for the executable part. Code is generated in either Java or PHP, object persistence is achieved on top of relational databases (Oracle, MySQL and PostGres).

Amazingly, the ability of being able to re-target different language and database combinations was of no interest to most commercial prospects. In practice, people were comfortable with a given technology mix and wanted to stay with that mix.

Dealing with Commercial Reality In 2000 the *.com bubble* burst and open-source development platforms such as Eclipse started to appear. Together, these factors made it impossible for ObjeXion to raise further development funding. The company managed to survive for two years by selling software development services that were realized with Netsilon. Several commercial applications were successfully developed one of which is still in production today (almost ten years later!).

Providing services was badly perceived by the team who viewed services as *the price one has to pay* for being able to develop the tool.

Shutdown Despite the new product and notwithstanding the fact that the technology provided realistic performance for real application development (as proven by several running commercial deployments) the company still had to face a lack of trust from potential customers. The key problem was that the company was trying to sell long term sustainability (the idea that models would be more stable in time than code), while being itself unstable (a common feature of startups). This created a chasm between the company value proposition and its perception by prospective customers. ObjeXion was able to find customers for professional services (performed using its own technology) but was not able to convince others (such as software houses) that using the technology would bring sufficient benefits (as compared to more traditional development techniques).

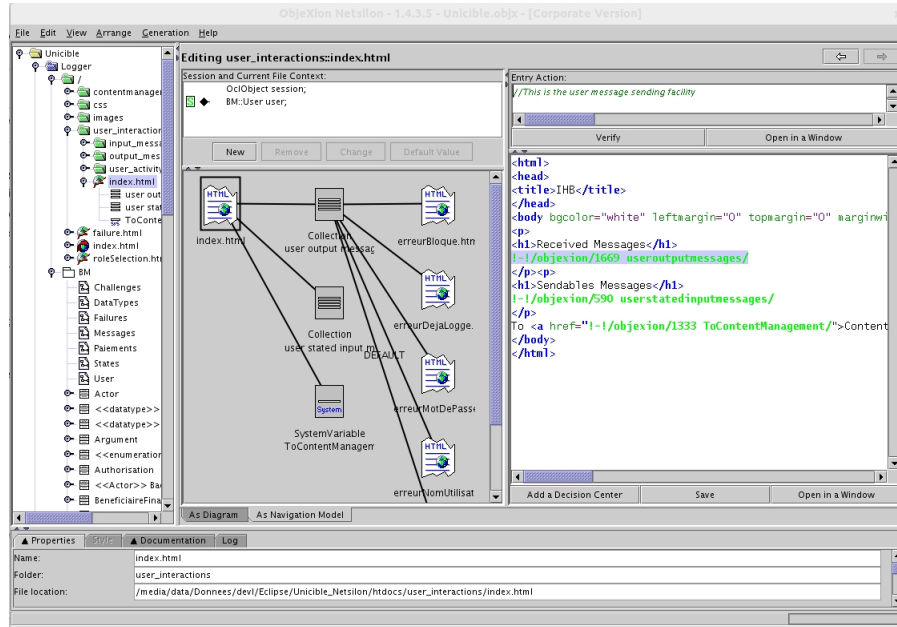


Fig. 3. Netsilon Tool

Pierre-Alain Muller went back to academia, and initiated the development of Kermeta (an open-source effort to build an executable meta-modeling language and dedicated environment, with a lot of similarities to XMF from Xactium) [25,26,17]; he is currently professor of Software Engineering, and involved in a university spin-off which is not using model-driven technologies.

Two other developers joined academia and are currently associate professors, still involved in model-driven matter. One developer became teacher of electrical engineering at high-school. The remaining engineers do not work in the MDD area.

Conclusion ObjeXion missed several critical junctures as described in section 3.1. First, the *opportunity recognition* was over-estimated, but as *entrepreneurial commitment* was very high, the founder was able to convince other people (including seed capitalists) to provide early stage funding. Over time it became obvious that the value proposal was unclear. As cash was still available, the company then identified another *opportunity* and basically started another business plan. The *threshold of credibility* was never reached. ObjeXion then entered into a survival mode, providing professional ser-

vices (which was basically yet another business model).

4 Lessons and Observations

This section outlines some lessons and observations arising from the authors' experiences described in the previous section. Section 4.1 describes some issues that, although general, are essential and can easily be ignored when establishing a high-tech startup. Section 4.2 describes issues that are specific to the MDD experiences of the two companies.

4.1 General Issues

The following general issues should be at the heart of any high-tech startup decision-making:

1. *You might be right - but who cares?* It is no good having the highest possible technical standards and knowing how to do something the right way, if no-one will listen. Learn to compromise.
2. *Leaping too soon.* It is easy to get caught up in the moment and drop everything else in order to create a high-tech startup. More often than not, this is a mistake.

3. *Business is commercially not technically driven.* Technical founders tend to forget that their company needs to make money, not technical innovation.
4. *One swallow does not make a summer.* Just because you have a single initial sale does not make it certain you will get many more.
5. *Think business development.* Institutional seed funding is very focussed on technical innovation, while venture capitalists are looking for business. Don't burn all your seed funding in technology, develop a business strategy and act on it as early as possible.
6. *The Swiss Army Knife.* A commercial product that does one thing well beats a product that does everything badly. Simplicity is a key feature that helps to sell technology - look at Google.
7. *Tool vendors cannot appear too small.* If a company is going to invest in a development tool then they need to be reassured that the company will exist in 6 months time.
8. *Tool development is expensive.* However much you think it will cost in terms of time and effort to develop a business based on a modelling tool, multiply by 10. Be prepared to be patient and support development through other activities.
9. *Don't sit in a dark room.* A typical problem with developers is that they want to continually develop technology. Stop early and sell what you have. Get out to the conferences and evangelize.
10. *Don't sell the next product.* A typical problem with sales-people is that they always want to sell the *next* version of the product. Sell what you have, not what would be nice to have.
11. *Be hungry.* All the major players in a company must be absolutely committed to its success. A major technical or commercial lead who lacks the hunger can easily sink a company.
12. *Don't put all eggs in one basket.* It is easy for a technical startup to construct initial designs and to continually develop a product based on the requirements of a single customer. Don't (unless your exit strategy depends on it).
13. *Sell Solutions* It is hard to sell technology and especially hard to sell new technology. It is much easier to sell business solutions that lead to a competitive edge.

4.2 MDD Issues

Both companies described in this article can be viewed as significant technical success stories. They managed to put together successful teams, raise development money, develop industrial-scale software tools to support leading-edge MDD techniques, and to support themselves via commercial activities based on the tools.

However, both companies failed to move from startups to established commercial vehicles. Some of the reasons for this failure are common to startups and are listed above. This section describes some of the technology-specific issues that led to failure.

1. *UML Usage* The current state-of-the-art in industrial MDD ranges from informal sketch-based modelling to the generation of code skeletons from blueprints. Practitioners are often sceptical about UML CASE tools and their ability to generate complete applications. It is not currently realistic for a small company to make grand claims about the ability of MDD to deliver large-scale business value in the general case.
2. *Investor Confidence* Investors need to know that there is a significant business that will arise from the use of MDD. Industry scepticism and the lack of high-profile business cases makes it difficult for MDD startups to attract seed funding.
3. *Novelty* MDD approaches are still considered novel and this raises a significant barrier for adoption by large companies. In most situations companies will trust the technologies that they are familiar with.
4. *Maturity* Industry would like to use MDD as a shrink-wrapped black-box process. Current technologies expose a great deal of the inner workings of PIM, PSM and transformation design. Developers feel that they need to have a detailed knowledge of all aspects of the technology which undermines its commercial value compared to the use of more trusted mature technologies such as compilers.

Model management is a problem for large-scale MDD adoption where multi-developer distributed projects are a requirement. Both companies described in this article undertook a major rewrite of their technology. This is typical of advanced technology platforms and is very difficult to achieve success-

- fully where existing customers have large model repositories.
5. *Opinions* It is hard to challenge preconceived opinions about MDD. In our experience, some industrial uses of MDD are badly performed and lead to misjudged opinions. This is perhaps related to the immaturity of MDD and the high-levels of technical expertise required to use tools. Furthermore, attempts to take corrective action through training and mentoring can have a negative effect due to increased use of resources.
 6. *Familiarity* Experience suggests that Software Engineers can feel trapped by MDD tools, which they feel compare unfavourably with their favourite development technologies. Software Engineers like to program not to model.
 7. *Text vs Diagrams* There is a large amount of evidence that software engineers prefer textual representations for system artifacts rather than diagrams. Even when generating programs from models, the temptation to edit the resulting code is overwhelming. This places MDD tools that are unable to cope with the scale of the changes, in an unfavourable light.
 8. *Platform Independence* MDA relies on developing platform independent models. A lack of consensus regarding what this term means in any given context can undermine confidence in MDD. Practitioners often argue that the risk associated with generating code from platform independent models is too great and of limited practical value. Where consensus is reached, it is often the case that PIMs include many aspects of the target platforms in order to support code generation.
 9. *Relational vs OO* Database administrators will often complain about the impedance mismatch between a relational and an OO model [22].
 10. *Maintenance* Once multiple models and associated code have been generated there is often a maintenance problem that is cited by companies as a reason for not adopting MDD. Multiple versions of UML serialization formats and a general lack of viable interoperability between MDD tools is cited as a business risk.
 11. *Complexity* MDD is often seen by industry as too heavyweight and complex. Problems arising from early adoption can often be perceived to exist long after technologies have matured. This can be seen in the failure of Ada where the first compilers were unusably slow, a perception that lasted long after compiler technology reached acceptable levels of maturity.
 12. *Generated Code* Experience suggests that industrial uses of MDD involving code generation, equates the quality of the technology with the readability of the generated code. Perhaps this arises from a lack of confidence with MDD where developers are particularly risk-averse. Opinions differ regarding the effectiveness of code generation [12]
 13. *Run-Time Models* Problems with code generation can be avoided by using run-time models. However, this generally involves some form of run-time model framework, possibly running on a server. Experience suggests that companies find it unappealing to install and manage general run-time model frameworks.

5 Conclusion

Traditional development approaches place restrictions on the developer by providing predefined features that must be used as part of the system description. For example: runtime behaviour may be handled by the operating system (priorities, scheduling, etc.), persistence may be driven by the database capabilities (locking granularity, transactions, etc.), types (whether static or dynamic), concrete syntax (and syntactic sugar) is predefined by the programming language, and cannot be altered.

Traditional environments enforce implied design decisions, letting developers focus on application functionalities. While this is good for some aspects of system development, it is also restrictive, in the sense that developers do not have complete flexibility about the way they package and factorize features.

Traditional programming languages have a wide general purpose spectrum, and tools (such as model checkers) are unaware of business logic; and therefore remain restricted to checking language constructs.

Reusable components are typically supported by language libraries (frameworks) and extensions remain horizontal (they are restricted to the M1 level); programmers import libraries, but cannot factor out features directly to

the programming languages (adapting the M2 level).

Xactium and ObjeXion shared a vision of explicit modeling. Explicit modeling (or explicit meta-modeling) means that all decisions are explicitly described by a modeling artifact. Xactium and ObjeXion postulated that programming with third-generation languages along with frameworks could and should be replaced by model-driven technology for real-world application development.

ObjeXion was to some extent more conservative than Xactium. ObjeXion addressed M2 technology, using a 3 GL (Java) for implementation. Xactium pushed the approach to the limit, and managed to bootstrap their own technology.

ObjeXion and Xactium made comparable mistakes. They were developing elegant tools for researchers, not pragmatic tools for engineers. Both companies were initiated by researchers who misunderstood that elegance does not necessarily lead to business. The people driving these two companies were confused by their own faith in novelty, backed up by their uncommon skills in modeling and language engineering, that unfortunately made them unaware of the overall complexity of the approach they were developing. They did not realize the gap between the current state of practice (programming with 3rd GLs and frameworks), and the state of the art modeling approach they were proposing. Neither did they realize that their approach would not scale in terms of manpower: there were not enough experts around in industry prepared to deploy sufficient copies of their products to make Xactium and ObjeXion profitable.

Taking aside their model-driven flavour, Netsilon and XMF-Mosaic were tools in competition with many other tools and development approaches. Netsilon and XMF Mosaic were doing nothing that could not be done by conventional techniques. These model-driven tools did not provide a real break-through, so that practitioners would abandon their current programming tools.

Model-driven development is an alternative to conventional development. Programming with 3rd GLs and frameworks has reached a form of consensus among practitioners, probably because it provides a good balance between expressiveness and precision, between good practice enforcement and design freedom; it can therefore be considered as an optimum. Model-driven

techniques from Xactium and ObjeXion challenged this optimum, raising the question of whether this optimum was local or global.

About ten years later, a conclusion could be to say that the model-driven kind of thinking, the idea of separating concerns such as with aspect oriented programming, the use of neutral common representations (such as XML and XML schemas), generative programming, and language annotation, altogether have brought the essence of meta-modeling in a pragmatic fashion in the world of 3rd GL. From this prospective, the spirit of model-driven technology is very alive, although absorbed by mainstream programming environments; this might be the price for success!

ObjeXion and Xactium have been victims of their over-optimistic *pure* point of view. Like many pioneers, they died with an arrow in their back. The next generation of companies making use of model-driven technologies might be more successful if they manage to hide model-driven technology, embedding it as a competitive advantage.

ObjeXion and Xactium were built as tool companies: companies whose business was to be built on top of some technological assets (in the current case, MDD technologies). Such a business model requires significant investment in marketing, sales, and support, which are way ahead of the initial cost for developing technology. There was not enough revenue potential in those single-tool based companies to be able to attract the required level of investment.

It is therefore questionable whether this kind of business model was even realistic at the time the companies were created (it was a sustainable model in the 80s, before the advent of Internet and global access to free software).

A business model based on selling tools is a difficult one, because:

- Buying software licences requires investment up-front from the buyer. The return on investment remains unclear for all but the most essential software engineering tools. Nobody really wants to pay for software licences any-more. The current trend is SaaS (Software as a Service) where software is seen as a commodity.
- Tools (such as XMF-Mosaic and Netsilon) do not produce end-user value that can easily be measured. The learning-curve must be balanced with productivity gains.

- Sales cycles are long, because several stakeholders must be convinced. Tools are usually not bought by experts, such as people in software houses (who only want to sell manpower), but by end-users, who do not expect to enter in technical discussions about potential benefits of using a development tool. Budget holders are rarely technical experts who can appreciate the benefits offered by technical excellence.

In the current global settings, with everything connected to everything else via the Internet, with high quality development environments available as free and open-source tools, with nobody wanting to pay for tools any-more, with cheap outsourced development capabilities, the way for a tool company, and more specifically a model-driven tool company, seems to be very narrow. The approach to business: *if you build a better mousetrap the world will beat a path to your door* seems inappropriate in the context of software development tools that are essentially free. It is therefore probably better to focus on end-user added value delivery, eventually realized with model-driven technologies, or with 3rd GL environments.

Building and selling software engineering tools has always been a hard business; it is technically complex, economically risky, and requires significant funding long before sales can generate substantial revenues. It is very difficult to measure the return on investment in software engineering, because quite often the return has to be considered beyond the scope of a single project. Therefore, investing in software engineering becomes a corporate decision, and this raises all kinds of new concerns, such as long term visibility, and also balance between business and technology (which is likely to be in favour of business, because the return will be quicker). As a conclusion, software engineering makes more sense for large companies developing product lines, over long period of times. Alas, large companies are not used to working with small new players, such as software engineering startups where it may take years before the small business becomes stable and profitable.

Obviously, MDD tools belong to this category, and have had to face this market reality. During the last decade, MDD has not become a mainstream technology. Some MDD tools managed to survive by supporting specific niche markets,

such as model-based testing (generating tests from models, instead of generating systems), or software modernization (easing translation of legacy applications into new technology).

The grand challenge of replacing programming by modeling has not been achieved. However, it looks like some advanced modeling techniques (such as model composition, or meta-modeling) have permeated the programming language sphere (think about aspects and annotations in Java). So a conclusion could be that modeling failed as a standalone approach but succeeded at extending the dominant 3rd generation language paradigm.

New trends in software engineering such as cloud computing and software as a service (SaaS) may find cases for MDD (probably in the area of automated configuration management, which comes back to product lines), if they manage to increase their perceived value by providing vertical solutions, which clearly solve a given problem. Traditional tools are by nature horizontal; they do not address specific needs (they are multi-purpose, very similar to Swiss-army knives) and this is why their perceived value is low.

Although commercial MDD tool development and adoption seems to have failed, the use of model-driven technology *as part* of commercial development seems to be thriving. In addition to niche application areas and recent language extensions such as Java annotations mentioned above, many applications can be populated using data (often in the form of XML), business process engines abound, database and content management systems are defined using models, and SaaS platforms provide users with interfaces that essentially build and subsequently process application models.

Therefore, model-driven approaches are far from obsolete and, despite all the troubles that we encountered building these two startup companies, it was an exciting, challenging and enlightening experience: *the best of times*.

References

1. D. Amyot, H. Farah, and J.F. Roy. Evaluation of development tools for domain-specific modeling languages. *System Analysis and Modeling: Language Profiles*, pages 183–197, 2006.
2. D.G. Bobrow, L.G. DeMichiel, R.P. Gabriel, S.E. Keene, G. Kiczales, and D.A. Moon. Com-

- mon lisp object system specification. *ACM Sigplan Notices*, 23(SI):1–142, 1988.
3. T. Clark, A. Evans, and S. Kent. The metamodelling language calculus: foundation semantics for UML. *Fundamental Approaches to Software Engineering*, pages 17–31, 2001.
4. T. Clark, A. Evans, and S. Kent. Engineering modelling languages: A precise meta-modelling approach. *Fundamental Approaches to Software Engineering*, pages 242–260, 2002.
5. T. Clark, A. Evans, and S. Kent. A metamodel for package extension with renaming. «UML» 2002, *The Unified Modeling Language*, pages 305–320, 2002.
6. T. Clark, A. Evans, and S. Kent. A programmers guide to MMT: <http://eprints.mdx.ac.uk/6280/1/ProgrammersGuideToMMT.pdf>, 2002.
7. T. Clark, A. Evans, and S. Kent. Aspect-oriented metamodelling. *The Computer Journal*, 46(5):566, 2003.
8. T. Clark, A. Evans, S. Kent, S. Brodsky, and S. Cook. A feasibility study in rearchitecting UML as a family of languages using a precise OO meta-modeling approach. *Report, Precise UML Group*, 2000.
9. T. Clark, P. Sammut, and J. Willans. Applied metamodelling: A foundation for language driven development. 2nd ed. 2008.
10. T. Clark and J. Willans. Software language engineering with XMF and XModeler. In M. Mernik, editor, *Formal and Practical Aspects of Domain Specific Languages: Recent Developments*. IGI Global, 2012.
11. P. Cointe. Metaclasses are first class: The objvlist model. In *ACM SIGPLAN Notices*, volume 22, pages 156–162. ACM, 1987.
12. M. Fowler. What is the point of the UML? In Perdita Stevens, Jon Whittle, and Grady Booch, editors, *UML 2003 - The Unified Modeling Language. Model Languages and Applications. 6th International Conference, San Francisco, CA, USA, October 2003, Proceedings*, volume 2863 of *LNCS*, page 325. Springer, 2003.
13. N. Georgalas, M. Azmoodeh, and S. Ou. Model driven integration of standard based oss components. *EURESCOM Summit 2005-Ubiquitous Services and Applications*, 2005.
14. A. Goldberg and D. Robson. *Smalltalk-80: the language and its implementation*. Addison-Wesley Longman Publishing Co., Inc., 1983.
15. S. Helsen, A. G. Ryman, and D. Spinellis. Where’s my jetpack? *IEEE Software*, 25(5):18–21, 2008.
16. John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical assessment of MDE in industry. In *Proceedings of the 33rd International Conference on Software Engineering, ICSE ’11*, pages 471–480, New York, NY, USA, 2011. ACM.
17. J.M. Jézéquel, O. Barais, and F. Fleurey. Model driven language engineering with kermeta. *Generative and Transformational Techniques in Software Engineering III*, pages 201–221, 2011.
18. A.G. Kleppe, J. Warmer, and W. Bast. *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Longman Publishing Co., Inc., 2003.
19. I. Kurtev. State of the art of qvt: A model transformation language standard. *Applications of Graph Transformations with Industrial Relevance*, pages 377–393, 2008.
20. B. Langlois, C.E. Jitia, and E. Jouenne. DSL classification. In *OOPSLA 7th Workshop on Domain Specific Modeling*. Citeseer, 2007.
21. R.J. Mann and T.W. Sager. Patents, venture capital, and software start-ups. *Research Policy*, 36(2):193–208, 2007.
22. E. Marcos, B. Vela, and J.M. Caverio. A methodological approach for object-relational database design using UML. *Software and Systems Modeling*, 2(1):59–72, 2003.
23. S.J. Mellor. *MDA distilled: principles of model-driven architecture*. Addison-Wesley Professional, 2004.
24. C. Midler and P. Silberzahn. Managing robust development process for high-tech startups through multi-project learning: The case of two european start-ups. *International Journal of Project Management*, 26(5):479–486, 2008.
25. P.A. Muller, F. Fleurey, and J.M. Jezequel. Weaving executability into object-oriented meta-languages. In *in: International Conference on Model Driven Engineering Languages and Systems (MoDELS), LNCS 3713 (2005)*, pages 264–278. Springer, 2005.
26. P.A. Muller, F. Fleurey, D. Vojtisek, Z. Drey, D. Pollet, F. Fondement, P. Studer, and J.M. Jézéquel. On executable meta-languages applied to model transformations. In *Model Transformations In Practice Workshop*. Citeseer, 2005.
27. P.A. Muller, P. Studer, F. Fondement, and J. Bézivin. Platform independent Web application modeling and development with Netsilon. *Software and Systems Modeling*, 4(4):424–442, 2005.
28. Pierre-Alain Muller, Philippe Studer, and Jean Bézivin. Platform independent web application modeling. In *UML*, pages 220–233. Springer Verlag, 2003.
29. G.C. O’connor and M.P. Rice. Opportunity recognition and breakthrough innovation in large established firms. *California Management Review*, 43(2):95–116, 2001.
30. R. M. Soley. Model driven architecture: The evolution of object-oriented systems? In *OOTS*, page 2, 2003.

31. D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley Professional, Boston, Massachusetts, 2008.
32. A. Vohora, M. Wright, and A. Lockett. Critical junctures in the development of university high-tech spinout companies. *Research Policy*, 33(1):147–175, 2004.